CUDA 의 메모리 복사로 인한 성능 저하 연구

강지훈*, 이대원**, 강인성*, 유헌창*
*고려대학교 컴퓨터교육학과
**서경대학교
e-mail : k2j23h@korea.ac.kr, daelee@skuniv.ac.kr, iskang@korea.ac.kr, yuhc@korea.ac.kr

A Study on a Declines in Performance by Memory Copy in CUDA

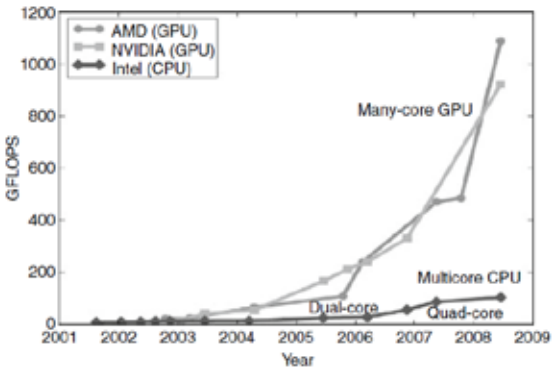
Jihun Kang*, DaeWon Lee**, InSung Kang*, HeonChang Yu*
*Dept. of Computer Science Education, Korea University
**Seokyeong University

요 약

GPGPU(General Purpose Graphics Processing Unit) 병렬처리 시스템인 CUDA(Compute Unified Device Architecture)는 컴퓨터에서의 고속 연산 처리를 위해 많이 사용되어왔다. CUDA 에서 연산 처리를 하기 위해서는 CUDA 의 특성을 이해해야 한다. CUDA 는 CPU(Central Processing Unit)가 처리하는 Host 영역과 GPU(Graphics Processing Unit)가 처리하는 영역인 Device 영역이 존재하며, 이 두 영역간의 데이터 복사를 통해 연산 처리를 진행한다. 이런 구조적인 특성상 메인 메모리에서 GPU 메모리로 입력 데이터를 전달해야 GPU 를 이용해 연산을 처리할 수 있는 구조를 가지고 있다. 하지만 이러한 처리 구조로 인해 연산 시간과 별도로 메인 메모리와 GPU 메모리간의 데이터 복사시간이 존재하며, 추가적으로 발생하는 메모리 복사 시간으로 인해 오버헤드가 발생하게 된다. 본 논문에서는 실험을 통해 메모리 복사 시간, 연산의 반복 횟수 그리고 연산의 복잡성이 전체 성능에 어떤 영향을 미치는지 논하고자 한다.

1. 서론

과거 GPU(Graphics Processing Unit)는 컴퓨터 그래픽스를 위한 연산처리를 담당하던 장치였다. 하지만 GPU 를 다양한 분야에 사용하기 위한 GPGPU(General-Purpose computing on Graphics Processing Units)의 개념이 나오면서 전통적으로 CPU(Central Processing Unit)에서 담당하던 연산을 CPU 와 비교하여 부동소수점 연산 능력이 뛰어난 GPU 를 사용하여 처리할 수 있었다. 그리고 GPU 는 다수의 코어로 구성되어 있기 때문에 수천 개의 스레드를 동시에 실행할 수 있다는 장점도 있었다. 그림 1 은 CPU 와 GPU 의 연산능력을 연도별 발전 현황과 함께 비교한 그림이다.



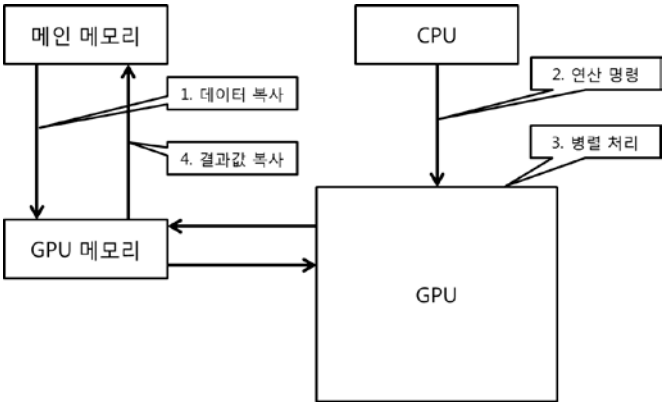
(그림 1) GPU 와 CPU 의 연산 능력 비교[1]

그림 1 에서 보여주는 것처럼 GPU 의 연산 능력이 CPU 의 연산능력보다 월등히 뛰어난 것을 알 수 있다. 연산능력이 뛰어난 GPU 에서 병렬처리를 통해 연산능력을 더욱 향상시키기 위해 개발된 기술이 CUDA(Compute Unified Device Architecture)[2]이다. CUDA 는 GPU 를 이용한 병렬처리 시스템으로써 대규모 연산을 고속으로 처리하는데 많이 이용된다. 이 기술은 GPU 에 존재하는 다수의 코어를 이용해 수천, 수만 개의 스레드를 동시에 실행시켜서 GPU 의 연산 능력을 극대화 시킬 수 있으며, 물리 계산, 암호학 등 다양한 분야에서 응용되어 연산처리에 대한 속도 향상을 가져왔다. 또한 C 언어, JAVA 등의 다양한 컴퓨터 언어에 사용할 수 있는 장점도 가지고 있다.

일반적으로 CUDA 응용 프로그램은 CPU 가 처리하는 영역인 Host 영역과 GPU 가 처리하는 Device 영역으로 분리된다. Host 영역에서는 입력데이터에 대한 메모리 할당 및 Device 영역에서 처리할 함수 등의 호출에 대한 작업을 하며[1], CPU 와 GPU 두 종류의 다른 코어를 사용하기 때문에 이종 컴퓨팅(heterogeneous computing)의 특징을 가지고 있으며, 일반적으로 프로그램의 전체 흐름은 CPU 에서 명령을 내리고 GPU 에서는 수학적인 연산에 대한 작업만 진행한다.

그림 2 는 CUDA 의 연산 처리 순서이다. 그림 2 에서 보여주는 것과 같이 CUDA 를 이용한 병렬처리는 크게 3 단계를 거친다. GPU 에서 연산을 처리하기 위해서는 메인 메모리에 작성된 입력데이터를 GPU 메

모리에 복사한 후, 복사된 데이터를 이용해 GPU 에서 연산에 대한 병렬처리를 진행한다. 그리고 마지막 단계로 GPU 에서 처리한 연산 결과를 메인 메모리에 복사를 하는 단계를 끝으로 CUDA 의 연산 사이클이 끝나게 되며, 모든 작업 명령은 CPU 를 통해 이루어진다



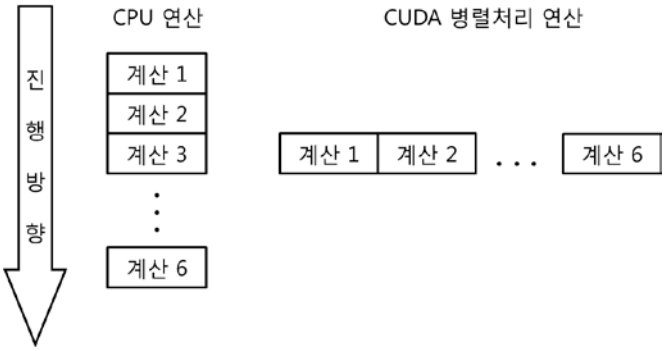
(그림 2) CUDA 의 연산 처리 순서[3]

CUDA 에서의 병렬 처리는 일반적으로 CPU 에서의 반복 연산을 병렬처리 하는 것이다. 예를 들어, CPU 에서 1+1, 2+2, 3+3……100+100 의 연산을 할 때, 일반적인 순차적 진행 방식에서는 각각의 덧셈 연산을 차례대로 100 회 반복해야 하지만 CUDA 는 병렬 스레드 100 개를 생성하여 한번에 처리한다는 것을 의미한다. 이러한 동시 처리는 큰 장점이 되며 반복 연산 횟수가 많을수록 처리 속도에 대한 이점을 얻을 수 있다. 이러한 장점이 있음에도 불구하고 CUDA 에도 단점이 존재한다. 그것은 CUDA 에서 병렬처리를 하기 위해서는 앞서 설명한 반복연산과 같은 처리를 해야 성능이 향상된다는 점이다. 즉 병렬처리를 하고자 하는 알고리즘의 처리 구조에 제약 점이 있다는 것이다. 그리고 CUDA 는 GPU 메모리에 데이터를 직접 쓰고 읽을 수 없기 때문에 메인 메모리에 데이터를 작성한 후 GPU 메모리에 복사를 해야 한다. 이러한 특징 때문에 메인 메모리와 GPU 메모리 사이에 데이터 복사 작업을 해야 하는데 이러한 과정에서 발생하는 메모리 복사시간이[4][5] 연산시간과는 별도로 추가로 발생한다. CUDA 에서 연산처리를 하게 되면 메모리 복사 시간은 큰 오버헤드로 작용한다.

본 논문에서는 실험을 통해 CUDA 의 단점인 메인 메모리와 GPU 메모리간의 데이터 복사 시간으로 발생하는 오버헤드에 대해 살펴보고자 한다. 또한 실험 결과를 통해 연산의 반복 횟수와 메모리 복사 시간이 전체 성능에 미치는 영향에 대해 논하고자 한다. 본 논문은 2 장에서 실험을 통해 메모리 복사시간으로 발생하는 성능 감소에 대해 살펴보고, 3 장에서 실험 결과에 대해 설명한다. 그리고 4 장에서 실험 결과를 통해 결론을 내린다.

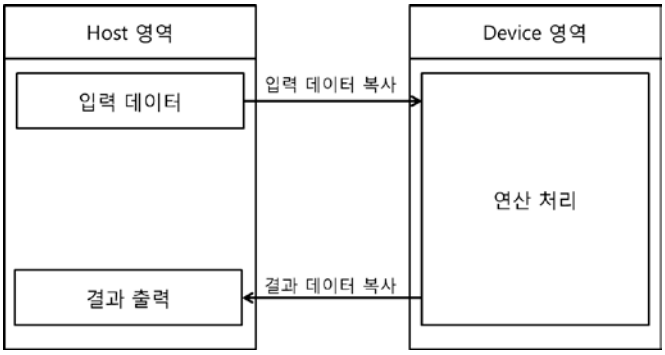
2. CUDA 를 이용한 병렬처리 성능 실험

앞에서 설명한 것과 같이 CUDA 에서의 병렬처리는 CPU 에서의 반복 연산을 동시에 처리하는 구조를 가지고 있다. CPU 를 이용한 계산과 CUDA 를 이용한 계산의 성능 차이는 GPU 와 CPU 의 연산 처리 능력의 차이도 있지만, 가장 큰 이유는 연산 처리 방식의 차이 때문이다. 일반적으로 순차 처리를 하는 CPU 는 반복 연산을 차례대로 진행하지만 CUDA 를 이용한 병렬 처리는 반복연산을 한번에 처리하게 되며, 이러한 차이 때문에 속도적인 성능 차이가 발생하게 된다. 아래 그림 3 은 CPU 와 CUDA 의 연산 처리 구조를 비교한 그림이다.



(그림 3) CPU 와 CUDA 의 연산 구조

본 논문에서는 2 개의 실험을 통해 CUDA 응용 프로그램에서 발생하는 메모리 복사시간과 프로그램의 성능의 연관성을 알아볼 것이다. 기본적으로 실험은 CPU 와 GPU 각각에서 동일한 연산처리를 실행하여 성능을 비교한다. 성능 비교를 위한 실험은 3 차원 좌표상의 벡터와 법선 벡터를 계산하는 2 개의 연산에 대한 실험을 통해 성능을 측정 할 것이다. 법선 벡터 계산은 벡터 계산의 연산 시간에 비해 상대적으로 시간이 많이 소요된다. 이 2 가지 실험을 GPU 와 CPU 를 이용한 연산 시간의 비교를 통해 연산에 소요되는 시간, 메모리 복사시간이 전체 성능에 미치는 영향을 알아볼 것이다. 실험에 추가할 다른 요소는 연산의 반복 횟수 증가이다. CPU 를 이용한 연산의 반복 횟수를 증가시키는 것은 CUDA 에서는 스레드를 증가시키는 것과 같은 의미이다. 실험에서 추가적으로 연산의 반복 횟수를 증가시키며 성능을 비교하여 연산의 반복 횟수가 성능에 미치는 영향 또한 알아볼 것이다.



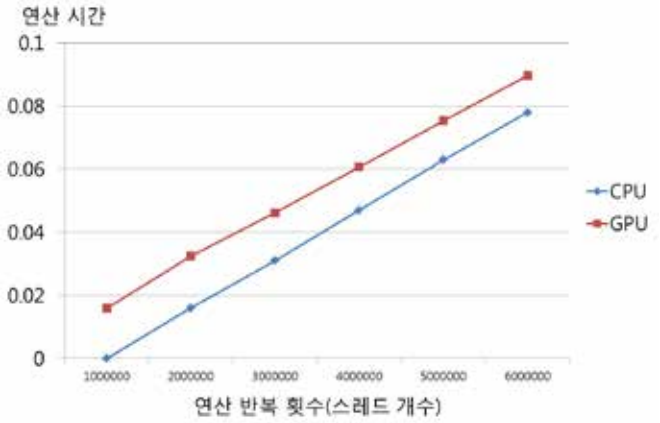
(그림 4) 프로그램 구조

실험에 사용된 프로그램의 전체 구조는 그림 4 과 같이 CPU 가 처리하는 영역인 Host 영역에서 입력 데이터를 작성한 후 GPU 에서 처리하는 Device 영역에 입력 데이터를 전달 받은 후 GPU 를 이용해 병렬처리로 연산 작업을 시작한다. 연산작업이 끝난 후 연산 결과를 Host 영역으로 복사하여 결과를 출력하는 일반적인 CUDA 응용 프로그램의 구조를 가지고 있다[5].

3. 실험 결과

이번 장에서는 CUDA 의 연산 처리에서 스레드의 개수, 메모리 복사 시간, 연산의 반복 횟수 그리고 연산의 복잡성이 성능에 미치는 영향에 대해 알아보기 위한 실험에 대한 결과를 설명한다. 본 논문의 실험은 Intel i5 3.2GHz, 4GB 의 메인 메모리, nVidia Geforce GTX 760, 윈도우 7 64 비트 운영체제의 환경에서 CUDA 4.0 버전을 이용해 실험하였다.

아래 그림은 GPU 와 CPU 에서 연산의 반복 횟수에 따른 벡터 연산의 시간을 비교한 그래프이며, 실험은 연산의 반복횟수를 순차적으로 1,000,000 번씩 증가시켰으며, 실험의 공정성을 위해 GPU 에 입력하는 데이터의 크기를 동일하게 맞춰 실험을 했다.



(그림 5) 벡터 계산 연산 시간 비교

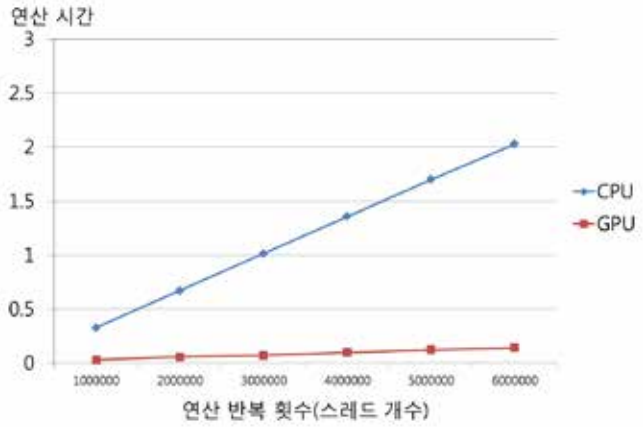
그림 5 의 그래프를 보면 알 수 있듯이 CPU 에서 6,000,000 회의 반복 연산을 GPU 에서 병렬처리를 하였지만 CPU 와의 성능을 비교했을 때 성능이 향상되지 않고 오히려 GPU 에서의 성능이 더 안 좋게 측정되었다. 그래프에서 보듯이 연산의 반복 횟수를 증가시키며 실험을 한 결과 GPU 에서의 처리가 CPU 에서의 처리보다 성능이 좋지 않지만 성능의 차이가 미세하게 좁혀지고 있다. 하지만 성능 차이가 줄어드는 감소 폭의 시간이 너무 미비하기 때문에 성능이 향상되고 있다고 보기 힘들다.

표 1 은 벡터 연산의 시간을 단계별로 분류한 표이다. 표 1 에서는 CUDA 와 CPU 의 연산 시간을 세부적으로 보여주고 있다. CPU 는 입력 데이터와 출력 데이터의 복사 시간이 존재하지 않기 때문에 작성하지 않았고, 각 부분의 연산 시간의 총 합이 전체 연산 시간과 같지 않은 것은 메모리 할당 및 해제 등 기타 작업에 소요된 시간을 표시하지 않았기 때문에 차이가 있는 것이다. 그리고 시간 표시는 소수점 넷째 자리까지 표시하였고 단위는 초이다.

<표 1> 벡터에 대한 세부적인 시간 측정 표

연산 횟수 (스레드 개수)	처리 장치	연산 시간			
		입력 데이터 복사	벡터 계산	출력 데이터 복사	전체
1,000,000	CPU	-	0.0000	-	0.0000
	CUDA	0.0069	0.0009	0.0040	0.0159
2,000,000	CPU	-	0.0000	-	0.0160
	CUDA	0.0138	0.0017	0.0091	0.0325
3,000,000	CPU	-	0.0150	-	0.0310
	CUDA	0.0213	0.0026	0.0117	0.0462
4,000,000	CPU	-	0.0160	-	0.0470
	CUDA	0.0275	0.0035	0.0154	0.0607
5,000,000	CPU	-	0.0160	-	0.0630
	CUDA	0.0349	0.0043	0.0197	0.0754
6,000,000	CPU	-	0.0310	-	0.0780
	CUDA	0.0419	0.0052	0.0234	0.0897

표 1 의 벡터 계산 시간을 보면 연산 반복 횟수가 2,000,000 까지는 CPU 에서의 연산 처리가 시간 표시 단위 미만으로 발생하여 CPU 에서의 성능이 좋았지만 3,000,000 회의 연산을 반복 할 때부터 CUDA 를 이용한 벡터 계산 시간이 약 6 배 정도 연산 시간이 단축되었으며, 그 이상 연산 반복 횟수를 증가시킨 실험에서도 약 4~6 배 정도 성능 차이를 보여주며 CUDA 에서의 처리가 더 빠르게 측정되었다. 하지만 GPU 에서 처리하기 위해 데이터를 복사하는 시간까지 감안하게 되면, 메모리 복사 시간이 계산시간 외에 추가로 소요되면서 전체적인 성능이 CPU 에서 처리하는 시간보다 성능이 좋지 않게 측정되었다.



(그림 6) 법선 벡터 계산 연산 시간 비교

그림 6 은 CPU 와 GPU 에서 각각 연산 처리한 법선 벡터 계산의 시간을 비교한 그래프이다. 법선 벡터 계산 실험의 경우 벡터 계산 실험과 동일한 방식으로 실험하였다. 그림 6 의 그래프를 보면 그림 5 의

그래프와는 다르게 GPU 를 이용한 연산이 CPU 에 비교하여 연산 시간에 대한 성능이 좋게 측정되었으며, 연산의 반복 횟수를 증가시킬 수록 연산 시간에 대한 성능이 크게 차이가 났다.

그림 6 의 그래프에서 보여주듯이 법선 벡터 계산에 대한 실험은 CUDA 를 이용하여 연산 속도 단축을 극대화 하는 예로 볼 수 있다. 입력 데이터의 크기가 벡터 계산 실험과 동일하다는 것을 감안했을 때, 법선 벡터 계산 실험의 경우 연산 속도에 대한 성능이 메모리 복사 시간보다는 연산의 복잡성에 대해 영향을 더 많이 받았다고 할 수 있다.

아래 표는 법선 벡터 계산에 대한 각 단계의 세부적인 시간 측정 표이다.

<표 2> 법선 벡터 계산에 대한 세부적인 시간 측정 표

연산 횟수 (스레드 개수)	처리 장치	연산 시간			
		입력 데이터 복사	벡터 계산	출력 데이터 복사	전체
1,000,000	CPU	-	0.3030	-	0.3280
	CUDA	0.0176	0.0021	0.0048	0.0312
2,000,000	CPU	-	0.6550	-	0.6860
	CUDA	0.0347	0.0041	0.0103	0.0610
3,000,000	CPU	-	0.9500	-	1.0300
	CUDA	0.0402	0.0062	0.0130	0.0754
4,000,000	CPU	-	1.3100	-	1.3720
	CUDA	0.0536	0.0082	0.0153	0.0980
5,000,000	CPU	-	1.6450	-	1.7160
	CUDA	0.0667	0.0103	0.0197	0.1238
6,000,000	CPU	-	1.9000	-	2.0280
	CUDA	0.0810	0.0123	0.0233	0.1482

표 2 를 보면 벡터 계산의 성능 비교 결과와 비슷하게 데이터 복사 시간이 실제 연산 시간보다 더 많이 소요되었다. 하지만 CPU 에서의 연산 복잡성이 증가하고 연산 반복 횟수가 많아지자 CUDA 를 이용해 연산을 처리한 결과에 비해 성능이 현저히 떨어졌으며 연산의 반복 횟수가 커질수록 연산 시간에 대한 성능 차이는 심해졌다.

4. 결론

본 논문에서는 2 개의 실험을 통해 연산의 반복 횟수, 메모리 복사 시간 그리고 연산의 복잡성이 CUDA 를 이용한 연산 처리의 성능에 어떤 영향을 미치는지 설명하였다. 벡터 계산의 실험 결과에서 보여주듯이 CUDA 에서의 연산 처리는 연산 횟수가 많아 지면 계산 시간에 대한 성능은 점점 향상되지만 메모리 복사 시간에 의해 전체 처리 시간은 CPU 보다 성능이 떨어지는 것을 볼 수 있었다. 이것은 연산 시간이 메모리 복사 시간에 비해 병렬 처리된 계산의 복잡도가 적었다는 것을 말한다. 반면, 법선 벡터 계산은 CPU 와 비교했을 때 CUDA 를 이용한 처리의 성능이 좋게 측정되었으며, 연산의 반복 횟수, 즉, 스레드의 개수가 증가할수록 성능 차이가 심해진 것을 볼 수 있다. 이것은 법선 벡터의 계산의 복잡도가 메모리 복사 시간을 충분히 수용할 정도로 많았으며, CUDA 를 이용한 연산 처리의 장점을 충분히 이용했다고 볼 수 있다.

본 논문에서의 실험을 통해 CUDA 의 연산 속도에 대한 장점을 얻기 위해서는 단순히 연산의 반복 횟수가 많은 것 보다 처리하고자 하는 계산의 복잡성이 더 큰 영향을 미치는 것으로 측정되었으며, 이러한 실험 결과는 CUDA 에서 존재하는 메모리 복사시간에 의해 발생 하였다.

참고문헌

[1] David B. Kirk, Wen-mei W. Hwu. “Programming Massively Parallel Processors”, Elsevier, 2010.
[2] http://www.nvidia.com/object/cuda_home_new.html
[3] http://ko.wikipedia.org/wiki/CUDA
[4] Sanders, Jason, Kandrot, Edward, “CUDA by Example”, Addison Wesley, 2010.
[5] CUDA C Programming Guide, nVidia, 2013